Методичка 2.4 - Анализ PE-файла с помощью OllyDbg

Компиляция программы на языке Си

Для проведения анализа PE-файла понадобится скомпилированная программа. Специально для этой цели была подготовлена программа prog-2.4.

Исходный код программы prog-2.4.c

```
#include <windows.h>

#pragma comment(lib,"user32.lib")

int say_zero(void);
int say_one_or_more(int);

int main(int argc, char* argv[]) {
	int a = argc;

	if (a == 1) {
		say_zero();
	}

	if (a > 1) {
		say_one_or_more(a);
	}
	return 0;
}

int say_zero(void) {
		char title[] = "First Function";
		char text[] = "Number of arguments: 0";

		MessageBox(NULL, text, title, 0);
		return 0;
}

int say_one_or_more(int tmp) {
		char title[] = "Second Function";
		char text[] = "Number of arguments: 1 or more";

		MessageBox(NULL, text, title, 0);
		return 0;
}
```

Компиляция:

> cl prog-2.4.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO

Описание параметров, которые были использованы при компиляции и линковке.

/GS- и /Gs-	- отключить защиту от переполнения буфера
/DYNAMICBASE:NO	- отключить механизм защиты ASLR
/NXCOMPAT:NO	- отключить механизм защиты DEP
/Od	- не выполнять оптимизацию кода
/link	- далее параметры для линкера

Как можно заметить, почти все параметры направлены на отключение различных механизмов безопасности. Они были отключены, для того, чтобы не усложнять код программы.

Запуск программы без отладчика с разным количеством входных аргументов.

> prog-2.1.exe

NoA: 0

> prog-2.1.exe 1

NoA: 1 or more

> prog-2.1.exe 1 2

NoA: 1 or more

Программа выводит количество входных аргументов.

Пролог и эпилог функции

```
; пролог функции
00401000    55        PUSH EBP
00401001    8BEC      MOV EBP, ESP

…

; эпилог функции
004010DB    8BE5      MOV ESP, EBP
004010DD    5D        POP EBP
```

Пролог - это набор инструкций, который выполняется в начале функции. Его основной задачей является создание стекового фрейма вызываемой функции.

Пример пролога:

```
00401000     55        PUSH EBP
00401001     8BEC      MOV EBP, ESP
```

Первая инструкция сохраняет в стеке текущее значение регистра EBP - push ebp. А вторая инструкция mov ebp, esp загружает текущий указатель стека в регистр EBP. В самом начале работы функции EBP и ESP равны.

Эпилог расположен последним в функции, его основной задачей является восстановление стекового фрейма для родительской функции.

Пример эпилога:

```
004010DB     8BE5      MOV ESP, EBP
004010DD     5D        POP EBP
```

Первая инструкция восстанавливает предыдущий указатель стека ESP, а вторая инструкция восстанавливает значение в регистре EBP.

Обзор программы под отладчиком OllyDbg

Запускаем отладчик OllyDbg и открываем программу prog-2.4.exe.

Первое, что нужно сделать, это определить местоположение. В заголовке окна указано, что мы в модуле prog-2_4.

Давайте перейдем в начало секции кода. Адрес секции кода можно узнать через окно - Memory (M).

Открываем окно Memory (M), находим в таблице секцию кода - .text для prog-2_4 и видим адрес - 0x00401000.

Переходим по этому адресу (Ctrl + G).

В самом начале секции расположен пролог какой-то функции. Судя по строке - “Number of arguments: 0” и вызову функции MessageBoxA, мы попали в функцию say_zero. В конце мы видим эпилог функции.

Спускаемся по коду ниже.

Далее мы видим функцию say_one_or_more.

Спускаемся по коду ниже.

Далее мы видим функцию main.

Давайте поставим точку останова (F2) в начале функции main - 0x004010D0 и продолжим выполнение программы (F9).

В начале мы видим пролог функции.

Далее мы получаем значение параметра argc и сравниваем его с 1.

Мы видим инструкции CMP и JNZ - это условный переход.

Затем идет вызов функции say_zero. Можем добавить комментарий - say_zero().

Вызываем функцию (F7).

В самом начале мы видим пролог функции.

Затем идет сохранение значений в регистрах ESI и EDI через стек. В конце функции они будут восстановлены.

Далее идет процесс получение строк из секции данных и запись их в стек через регистры.

Видим, что по адресу 0x413000 расположена строка - First Function.

Первые 4 байта загружаются в регистр EAX, а затем идет запись этих байт в стек.

Следующие 4 байта строки загружаются через регистр ECX. Следующие 4 байта через регистр EDX. Далее 2 байта через регистр AX. И последний нулевой байт через регистр CL.

Далее в регистр ECX загружается значение 5. Это счетчик для инструкции REP.

В регистр ESI загружается адрес начала строки.

В регистр EDI загружается адрес, куда будет записана строка.

Можем в окне Dump вывести область памяти, куда будет записываться строка.

Ctrl + G, EDI

Далее мы видим инструкцию REP которая копирует 4 байта из адреса в регистре ESI в адрес, который указан в регистре EDI. И делает это 5 раз. Получает за одну инструкцию будет скопировано 20 байт, т.е. Первые байт строки - “Number of arguments: 0”.

Затем с помощью инструкции MOVS копируются оставшиеся 2 байта строки.

Далее с помощью инструкции MOVS копируются нулевой байт.

Затем подготавливаются параметры и вызывается функция MessageBoxA.

После вызова функции обнуляется значение в регистре EAX, так как функция возвращает нулевое значение.

В конце мы видим, что значения в регистрах ESI и EDI восстанавливаются, а также эпилог функции.

После инструкции RET мы возвращаемся в функцию main. Второе условие у нас не выполняется и мы переходим к эпилогу функции.

Мы прошли только по одной ветке программы, чтобы пройти по другой, необходимо изменить количество входных аргументов.

Для того, чтобы задать входные аргументы для программы можно воспользоваться пунктом меню Debug - Arguments и перезапустить программу - Ctrl + F2.